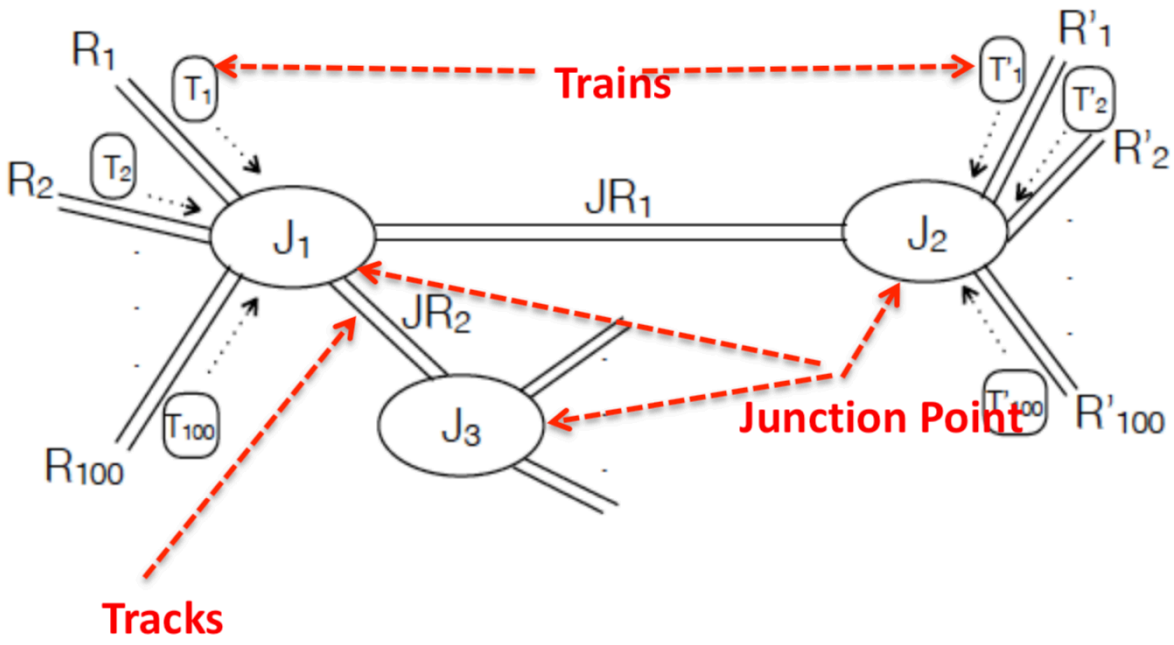
Problem Set 1

2020 02 20 16:40

(10 points) Cohort Exercise 1.2:

A railway network consists of several tracks, junction points and trains. A train moves along the track until it requires changing the track to reach destination. The changes of track occur only through a junction point. Each train is initially positioned at a junction point and its destination is a different junction point. Hence, while a train approaches its destination junction point, it does not need to change tracks any more. Each track can have at most one train at a time to avoid collision, but all tracks in the railway network are bi-directional. Design a set of APIs (application programming interface) for a safe (i.e. collision free) system. No coding is required, the APIs and their purpose will suffice.



<interface> Train
destination current_track_and_direction
change_track()

<interface> Track
connected_junction_points

connected_junction_points current_train_direction_1 current_train_direction_2
is_occupied()

<interface> Junction
connected_tracks

(10 points) Cohort Exercise 1.3:

Consider the railway network from previous exercise. Assume tracks of different type – broad gauge, meter gauge and narrow gauge. Each junction point is further divided into individual establishments that exclusively handle a specific track (i.e. meter, broad or narrow). Similarly, trains for meter gauge track is different from trains for broad gauge and narrow gauge. Narrow gauge trains are not powerful to run longer than the distance between two junction points. At each junction point, therefore, its engine is changed. Refine your set of APIs to capture this system.

<interface> Train
destination current_track_and_direction gauge
change_track() change_engine()

<interface> Track
connected_junction_points gauge current_train_direction_1 current_train_direction_2
is_occupied()

<interface> Junction
connected_tracks
list_all_broad_gauge_tracks() list_all_meter_gauge_tracks() list_all_narrow_gauge_tracks()

(10 points) Cohort Exercise 2:

Design and implement a program that supports accepting two complex numbers from the user; adding, subtracting, multiplying, and/dividing them; and reporting each result to the user.

complex_number.py

```
class ComplexNumber(object):
    real_part = 0.0
    imaginary_part = 0.0
    def __init__(self, real_part, imaginary_part):
        self.real_part = real_part
        self.imaginary_part = imaginary_part
    def __str__(self):
        # return "true"
        return f'{self.real_part} + {self.imaginary_part}i'
    def __add__(self, other):
        return ComplexNumber(self.real_part + other.real_part,
                             self.imaginary_part + other.imagi
nary_part)
    def __sub__(self, other):
        return ComplexNumber(self.real_part - other.real_part,
                             self.imaginary_part - other.imagi
nary_part)
    def __mul__(self, other):
        return ComplexNumber((self.real_part * other.real_part
 -
                              self.imaginary_part * other.imag
inary_part),
                             (self.real_part * other.imaginary
_part +
                              self.imaginary_part * other.real
_part))
    def __truediv__(self, other):
        return ComplexNumber((self.real_part * other.real_part
 +
                              self.imaginary_part * other.imag
inary_part) /
```

```
                                (other.real_part**
2 + other.imaginary_part**2),
                                (self.imaginary_part * other.real
_part -
                                 self.real_part * other.imaginary
_part) /
                                (other.real_part**
2 + other.imaginary_part**2))

import re
expression_pattern = re.compile(
    r'\(\d+(\+|\-)\d+[i]\)(\+|\-|\*|\/)\(\d+(\+|\-)\d+[i]\)')
operator_pattern = re.compile(r'\)(\+|\-|\*|\/)\(')
number_pattern = re.compile(r'\d+')
if __name__ == "__main__":
    input_string = input(
        "please input an expression to be evaluated, eg (1+
2i)*(2+1i):\n")
    if expression_pattern.match(input_string) == None:
        raise (
            ValueError("please input a correct expression, eg (
1+2i)*(2+1i)"))
    operator_span = operator_pattern.search(input_string).span
()
    operator = input_string[operator_span[0] + 1:operator_span
[1] - 1]
    numbers = number_pattern.findall(input_string)
    c1 = ComplexNumber(int(numbers[0]), int(numbers[1]))
    c2 = ComplexNumber(int(numbers[2]), int(numbers[3]))
    if operator == "+":
        print(c1 + c2)
    elif operator == "-":
        print(c1 - c2)
    elif operator == "*":
        print(c1 * c2)
    elif operator == "/":
        print(c1 / c2)
```

complex_number_test.py

```
import pytest
from complex_number import ComplexNumber

class TestComplexNumberSimple:
    c1 = ComplexNumber(1, 1)
    c2 = ComplexNumber(0, 0)
    def test_str_simple(self):
        assert str(self.c1 * self.c2) !
= str(ComplexNumber(0, 0)) + " "
```

```
    def test_addition_simple(self):
        assert str(self.c1 + self.c2) == str(ComplexNumber(1,
1))
    def test_subtraction_simple(self):
        assert str(self.c1 - self.c2) == str(ComplexNumber(1,
1))
    def test_multiplication_simple(self):
        assert str(self.c1 * self.c2) == str(ComplexNumber(0,
0))
    def test_division_simple(self):
        with pytest.raises(ZeroDivisionError):
            self.c1 / self.c2

class TestComplexNumber1:
    c1 = ComplexNumber(2, 1)
    c2 = ComplexNumber(3, -2)
    def test_addition_1(self):
        assert str(self.c1 + self.c2) == str(ComplexNumber(5,
-1))
    def test_subtraction_1(self):
        assert str(self.c1 - self.c2) == str(ComplexNumber(-1,
 3))
    def test_multiplication_1(self):
        assert str(self.c1 * self.c2) == str(ComplexNumber(8,
-1))
    def test_division_1(self):
        assert str(self.c1 / self.c2) == str(ComplexNumber(4 /
 13, 7 / 13))
```

```
pytest
============================================================
test session starts
============================================================
platform darwin -- Python 3.7.6, pytest-5.2.1, py-1.8.0,
pluggy-0.13.0
rootdir: /Users/ALEX/Documents/Term 5/50.003 Elements of
Software Construction/ps1
collected 9 items

complex_number_test.py .........
[100%]

============================================================ 9
passed in 0.02s
============================================================
```

sample usage

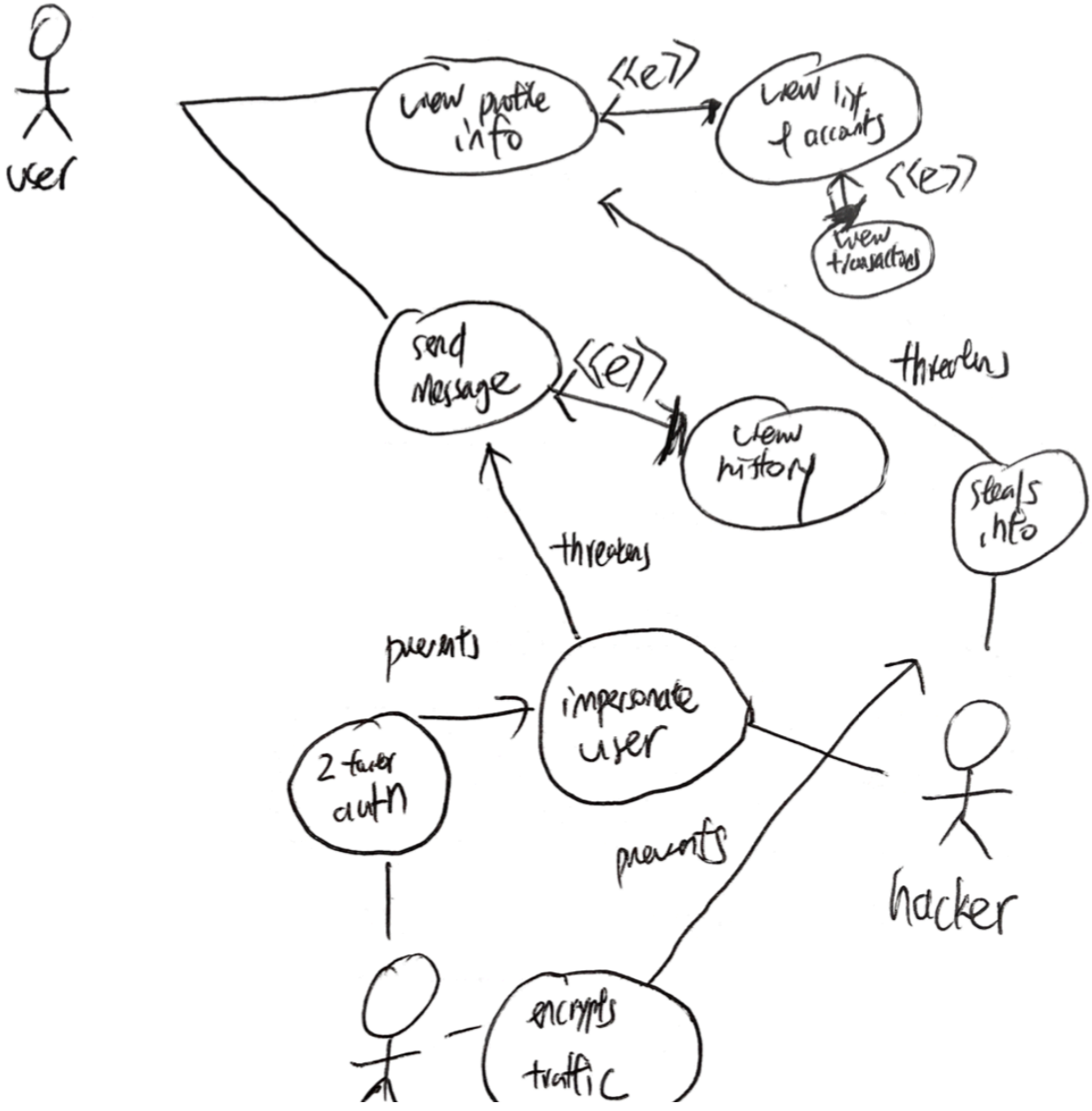
```
please input an expression to be evaluated, eg (1+2i)*(2+
1i):
(2+5i)*(2-4i)
-16 + 18i

(10-20i)/(15-30i)
0.6666666666666666 + 0.0i
```

(15 points) Cohort Exercise 4:

Draw a user case diagram for KBO (the online banking system discussed in the class).

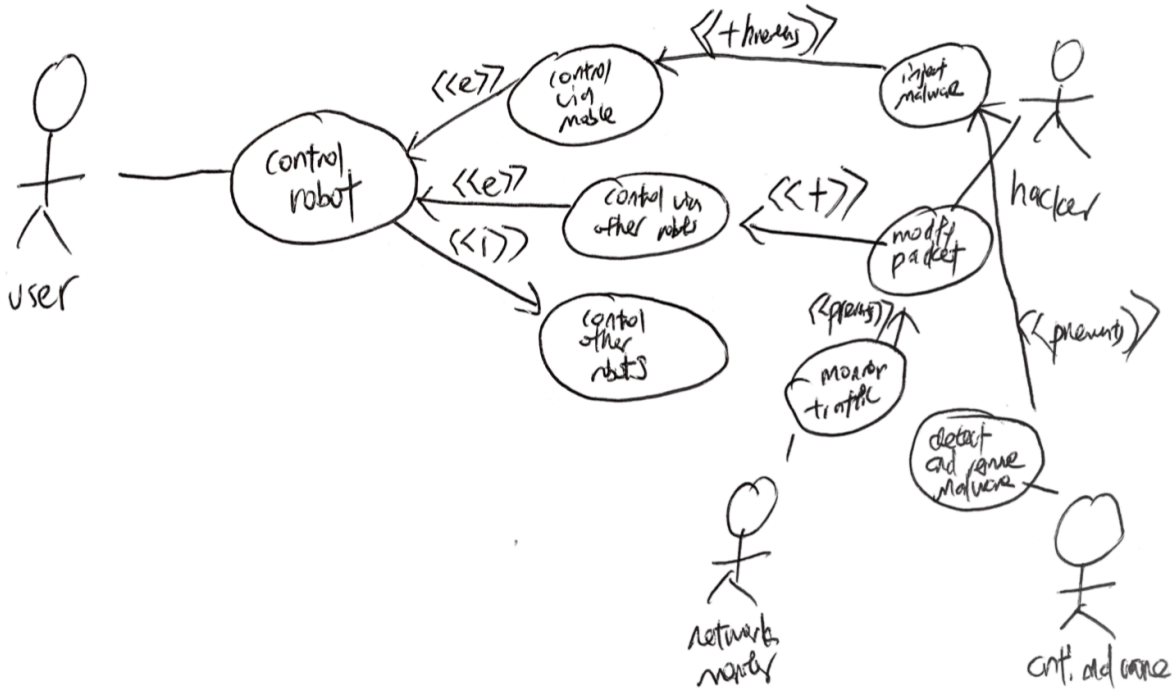
(15 points) Cohort Exercise 5:

Augment the use case diagram for KBO with at least two misuse cases and additional use cases to prevent the misuse.



(10 points) Cohort Exercise 6:

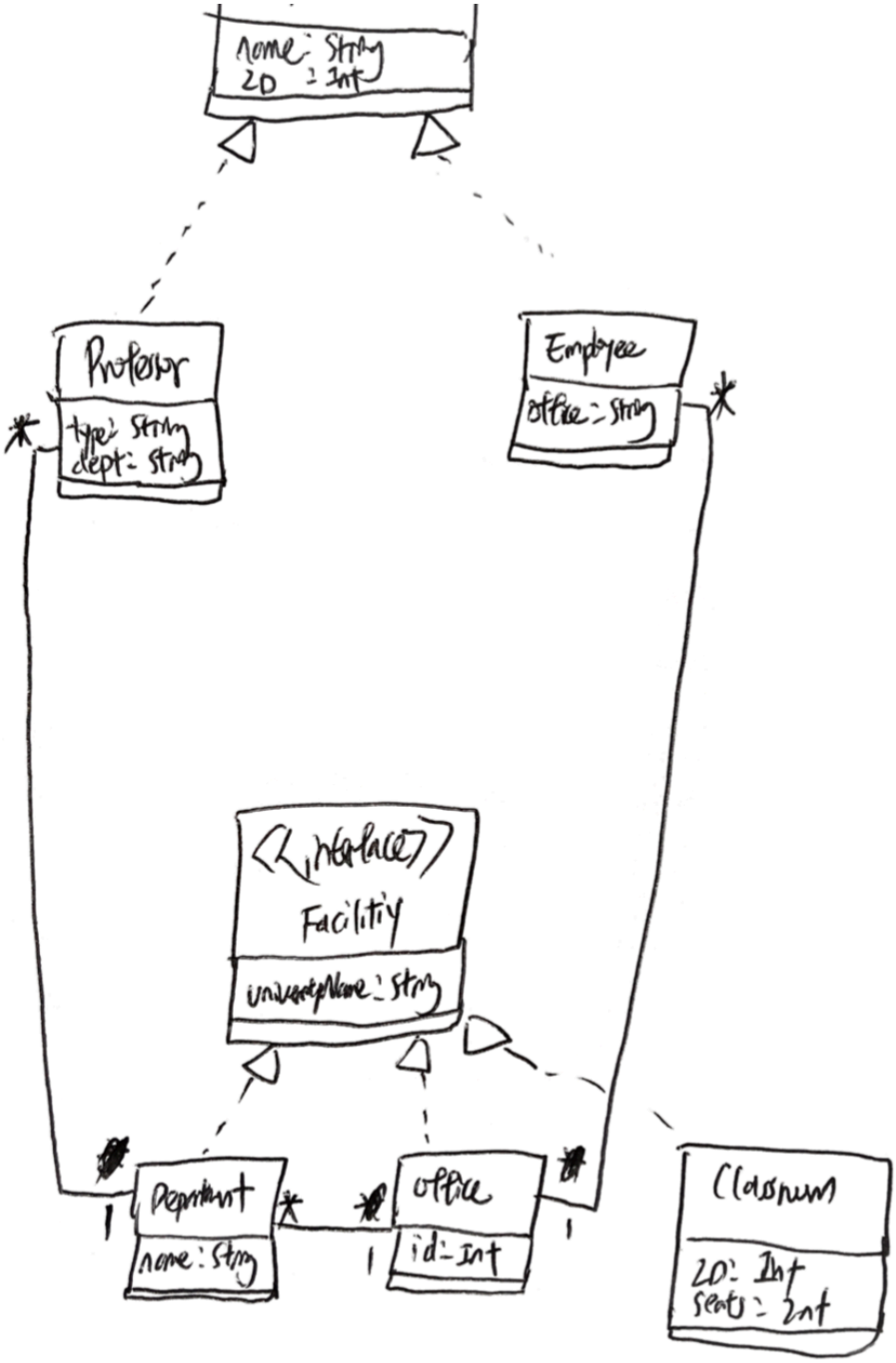
Consider a swarm of robots where a user can control any robot via mobile. Each robot can communicate to the user or to another robot. A hacker can affect the system by injecting a malware in the mobile or tapping on the communication between robots (e.g. modifying or delaying packets sent through the network). Draw the combined use-misuse case diagram for the system. Integrate anti-malware and network monitoring as security solutions for blocking attackers.



(10 points) Cohort Exercise 7:

Draw a class diagram for the following scenario. In a university there are different classrooms, offices and departments. A department has a name and it contains many offices. A person working at the university has a unique ID and can be a professor or an employee.

- A professor can be a full, associate or assistant professor and he/she is enrolled in one department.
- Offices and classrooms have a number ID, and a classroom has a number of seats.
- Every employee works in an office.



(20 points) Cohort Exercise 8:

A hardware update wizard can be in three states as follows:

1. Displaying a hardware update window.

2. Searching for new hardware.

3. Displaying new hardware found.

The wizard starts by displaying a hardware update window. While displaying this window, the user can press a "Search" button to cause the wizard to start searching for new hardware, or the user can press a

"Finish" button to leave the wizard. While the wizard is searching for new hardware, the user may cancel the search at any time. If the user cancels the search, the wizard displays the hardware update window again. When the wizard has completed searching for new hardware, it displays the new hardware found. Draw a state machine diagram that represents the function of the hardware update wizard just described.